Programación Python para Machine Learning

Lección 13: Deep Learning I.

eip ESCUELA INTERNACIONAL POSGRADOS

ÍNDICE

Lección 13: Deep Learning I.

1. Introducción

El Deep Learning representa en la actualidad una de las áreas más prometedoras dentro del Machine Learning y la Inteligencia Artificial. La aparición y perfeccionamiento de este tipo de modelos pone a disposición de los desarrolladores un abanico de nuevas posibilidades de aplicación para la resolución de problemas supervisados y no supervisados. A lo largo de esta lección se estudiarán los principios básicos de los modelos de Deep Learning, las principales diferencias con los modelos de Machine Learning clásicos, así como se presenta las numerosas, diversas y potentes bibliotecas de Deep Learning, tales como TensorFlow o Keras dentro de Python, lenguaje de referencia en Machine Learning.

Objetivos

- Saber qué es el Deep Learning y en qué situaciones puede ser útil.
- Identificar los requisitos software y hardware para desarrollar proyectos de Deep Learning.
- Conocer los distintos modelos de Deep Learning existentes.
- Dominar las técnicas de implementación de modelos de Deep Learning en Python.

2. ¿Qué es y cuándo utilizar Deep Learning?

El prometedor rendimiento de los algoritmos de Deep Learning en tareas de alta dificultad, unido a la creciente disponibilidad de modelos precomputados, han hecho que esta rama del Machine Learning resulte especialmente atractiva en los últimos años en la resolución de problemas dentro del ámbito de la Inteligencia Artificial.

Los actuales modelos de Deep Learning surgen y se empiezan a hacer populares gracias a la confluencia de tres aspectos:

- Los principios de los modelos de Redes Neuronales, existentes desde hace años, pero que sirven de base e inspiración.
- El desarrollo de bibliotecas muy optimizadas para poner en funcionamiento este tipo de modelos.
- La evolución de un determinado tipo de hardware (GPU y TPU).

Todo ello ha supuesto el impulso definitivo para modelos de Redes Neuronales con mucho más tamaño y profundidad, que parecen ser tremendamente útiles a la hora de resolver ciertos tipos de problemas, tales como el reconocimiento de objetos en imágenes, análisis de sentimientos, predicción de series temporales, reconocimiento de voz...

Sin embargo, esto no significa que el Deep Learning sea la respuesta a todos los problemas relacionados con el Machine Learning. Aunque cada problema tiene unas características individuales y dependerá de objetivos específicos, es fundamental saber cuándo es necesario la aplicación de técnicas de Deep Learning y cuándo no.

Una de las principales ventajas del Deep Learning está en poder resolver problemas complicados que requieren descubrir relaciones complejas ocultas entre un gran número de variables interdependientes en un conjunto de datos. Este tipo de modelos son capaces de aprender estas relaciones, combinarlas y construir reglas de decisión muy eficaces. Es decir, realmente son útiles cuando se trata de tareas complicadas, especialmente cuando se trata con muchos datos no estructurados.

Para tareas más sencillas, que implican un procesamiento de características más directa sobre conjuntos de datos estructurados, las técnicas de Machine Learning clásicas puede ser una mejor opción.

En los modelos de Deep Learning, se automatizan las tareas de ingeniería de características, evitando la intervención del programador en tal cuestión. Sin embargo, este hecho hace que se vea afectada la interpretabilidad del modelo resultante. Es una realidad que los modelos de Deep Learning alcanzan un rendimiento muy superior a la de los métodos clásicos de Machine Learning en determinados problemas, pero al modelar relaciones de alta complejidad no lineales, son casi imposibles de interpretar. Objetivamente, la interpretabilidad de sus modelos es uno de los mayores retos que presenta aún el Deep Learning.

A pesar de que cada vez son más accesibles, en la práctica la implementación y uso de modelos de Deep Learning sigue siendo un aspecto delicado por su dificultad y costo. Debido a su inherente complejidad por el número de capas y las ingentes cantidades de datos que se requieren, este tipo de modelos son muy lentos y requieren mucha potencia de cálculo. Es prácticamente obligatorio disponer de una infraestructura que dispongan de GPU para ejecutar dichos algoritmos, con el consecuente aumento del coste económico que ello conlleva, ya que la puesta en marcha de estas metodologías no sería viable.

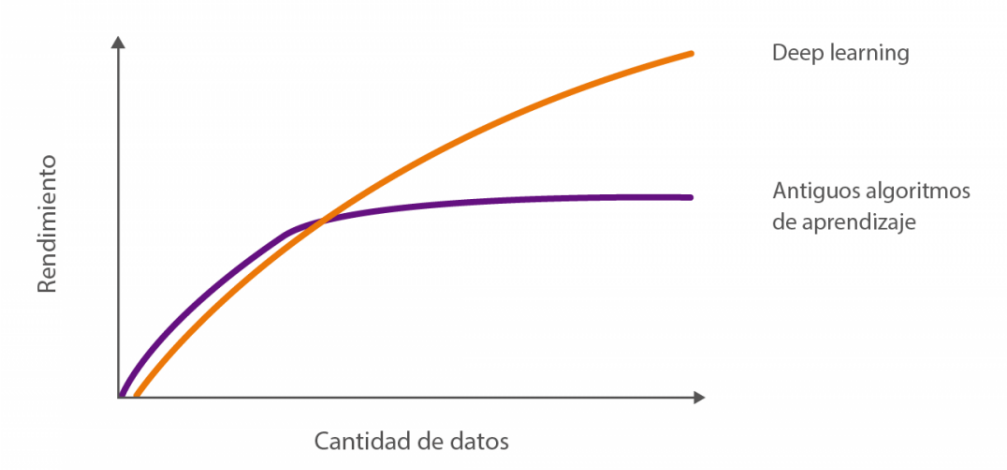

Ilustración 1: Rendimiento de los algoritmos de aprendizaje en función de la cantidad de datos.

Por último, los modelos de Deep Learning no pueden ser pensados como entes que aprenden por sí mismos. Este tipo de modelos siguen necesitando datos debidamente etiquetados. Uno de los principales puntos fuertes del Deep Learning es que pueden aprender relaciones más complejas, pero a costa de incrementar la complejidad de sus algoritmos. Para poder extraer estas relaciones, necesitarán muchas más instancias que un modelo de Machine Learning tradicional. La falta de un conjunto suficientemente grande de datos de alta calidad etiquetados con precisión es una de las principales razones por las que los modelos de Deep Learning puede tener resultados decepcionantes. En los casos en los que no es posible o necesario disponer de grandes cantidades de datos etiquetados de forma adecuada, los modelos de Machine Learning clásicos son la opción adecuada.

3. TensorFlow y Keras

TensorFlow es una biblioteca de código abierto para la computación numérica y Machine Learning a gran escala. Creada y mantenida por Google, reúne una serie de modelos y algoritmos de Machine Learning y Deep Learning.

Dispone, entre otras, de una API para Python. Fue diseñada para ser utilizada tanto para investigación y desarrollo como para poner sistemas en producción. Puede ejecutarse en sistemas con CPU, GPU, en dispositivos móviles o en grandes sistemas distribuidos de un elevado número de máquinas. No obstante, a pesar de su potencia, crear directamente un modelo a partir de TensorFlow puede resultar una tarea dificultosa.

La biblioteca Keras proporciona de modos más directos y limpios para generar numerosos tipos de redes profundas utilizando la base que presta TensorFlow.

Keras es una biblioteca de Python para crear modelos de Deep Learning de que corren sobre TensorFlow de manera rápida y fácil. Desarrollada y mantenida por François Chollet, ingeniero de Google, bajo licencia MIT, pretende ser una herramienta modulable, minimalista y extensible.

3.1. Instalación de TensorFlow y Keras

Existen múltiples maneras de instalar TensorFlow y Keras. Las maneras más populares son utilizar la gestión de paquetes del propio sistema operativo, utilizar la herramienta de gestión de paquetes de Python *pip* o hacer uso de Anaconda. La documentación tanto de TensorFlow como de Keras es excelente y cubre las instrucciones de cómo realizar el proceso sobre distintas plataformas. Facilita mucho la tarea el hecho de que la instalación de TensorFlow 2.0+ por defecto también instala Keras.

Una vez instaladas ambas librerías para Python, se debe confirmar que la instalación se ha realizado correctamente. Para ello, es necesario ejecutar el siguiente código de Python para imprimir las versiones de las bibliotecas instaladas:

```
import tensorflow
print(tensorflow.__version__)

from tensorflow import keras
print(keras.__version__)
```

4. Hardware necesario para Deep Learning

Los modelos de Deep Learning suelen ser por lo general tremendamente exigentes desde el punto de vista computacional. Aunque pueden ser ejecutados en CPU, el tiempo de ejecución puede ascender a horas, días, semanas para obtener un resultado, incluso que directamente el tiempo de cómputo sea inadmisible para el proyecto. Si se tiene la posibilidad de ejecutar los modelos de Deep Learning sobre sistemas que cuenten con GPU, el tiempo de entrenamiento se puede acelerar drásticamente. En este sentido, es posible acceder a servicios que ponen a disposición infraestructuras GPU para proceder con la ejecución de modelos de Deep Learning. Por ejemplo, Amazon Web Service (AWS), por menos de un euro por hora puedes crear y utilizar instancias de servidor virtual EC2 en la que ejecutar Keras.

Por otro lado, Google ha desarrollado un nuevo tipo de unidad de procesamiento denominada TPUs, unidad de procesamiento de tensores. Su arquitectura ha sido específicamente diseñada para la ejecución de modelos de Machine Learning, puesto que está optimizada para proceder con operaciones matriciales en paralelo y durante el proceso no se requiere acceso a memoria en ningún momento. Actualmente, las TPU es la arquitectura más potente para este tipo de algoritmos. Google Colab es un producto de Google Research que permite a cualquier usuario escribir y ejecutar código Python. Es especialmente adecuado para tareas de Machine Learning y análisis de datos. Desde un punto de vista más técnico, Colab es un servicio alojado de Jupyter Notebook que no requiere configuración y que ofrece acceso gratuito a recursos (limitados) computacionales como GPUs o TPUs.

5. Tipos de modelos de Deep Learning

Los modelos de redes neuronales de Deep Learning contienen múltiples capas ocultas entre las capas de entrada y la de salida. Si N es el número de capas ocultas, una estructura general de una red neuronal profunda tiene N ≥ 2, al contrario de lo que ocurre con las redes neuronales clásicas o superficiales donde N=1.

Dicho esto, se podría componer una taxonomía de modelos de Deep Learning como la mostrada en la Ilustración 2 si se dividen las técnicas en dos grandes categorías según la naturaleza del problema:

1. Redes profundas para el aprendizaje supervisado.

   Redes Neuronales Profundas (DNN) son modelos de redes neuronales con múltiples capas ocultas completamente conectadas.

   Redes Neuronales Convolucional (CNN o ConvNet) son una popular arquitectura de Deep Learning supervisado que aprenden directamente de la entrada sin necesidad de extraer características por parte de los humanos.

   Redes Neuronales Recurrentes (RNN) también es un tipo de red neuronal muy popular, que emplea datos secuenciales o de series temporales y alimenta la salida de la etapa anterior como entrada a la etapa actual. Han surgido diseños de diferentes tipos:

   - Memoria a largo corto plazo (LSTM) son una forma popular de arquitectura RNN que utiliza células de memoria para almacenar datos durante largos periodos y el flujo de información dentro y fuera de la célula se gestiona mediante tres puertas.

- Las RNN bidireccionales conectan dos capas ocultas que van en direcciones opuestas a una única salida, lo que les permite aceptar datos tanto del pasado como del futuro.
- Una Unidad Recurrente Controlada (GRU) es otra variante popular de la red recurrente que utiliza métodos de compuerta para controlar y gestionar el flujo de información entre los nodos de una red neuronal.

Redes de creencia profunda (DBN) constan de varias capas que son máquinas restringidas de Boltzmann, RBM, y una capa de clasificación softmax.

2. Redes profundas para el aprendizaje no supervisado.

Una máquina de Boltzmann restringida (RBM) es una red dos capas, capa de entrada y capa oculta, que aprende una distribución de probabilidad basada en el conjunto de entradas.

Redes generativas antagónicas (GAN) son modelos en los que dos redes neuronales compiten, una generadora y otra discriminadora. La red generadora intenta crear conjuntos de datos que tengan una apariencia tan auténtica que engañen a la discriminadora.

Un Autoencoder es una popular técnica que utiliza redes neuronales para aprender representaciones de los datos para reducir la dimensionalidad. Tiene tres fases: codificador, código y decodificador. El codificador comprime la entrada y genera el código, que el decodificador utiliza posteriormente para reconstruir la entrada.

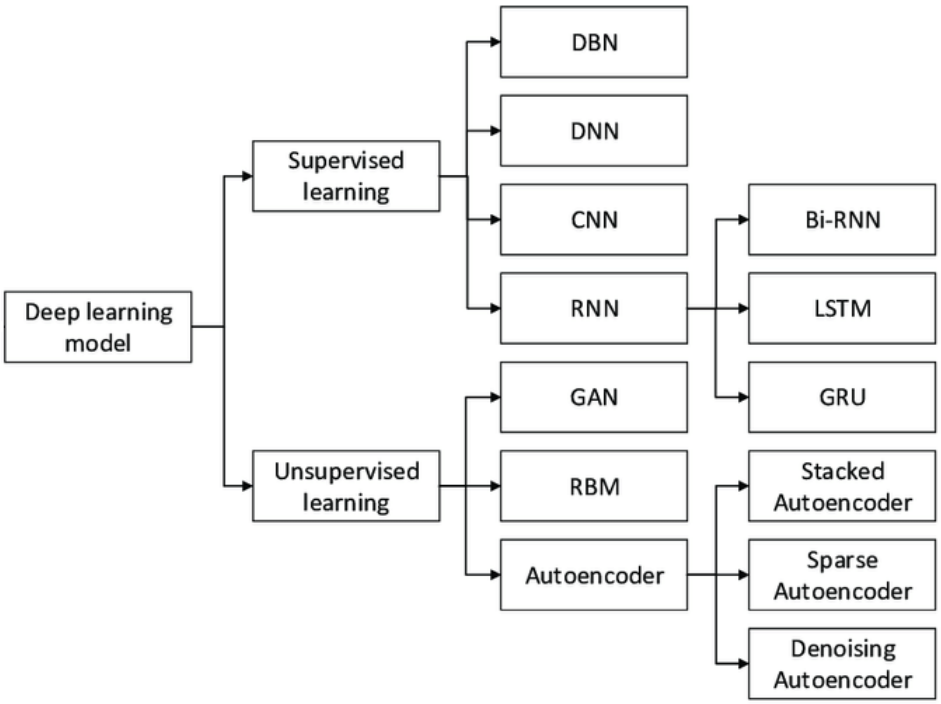

Ilustración 2: Taxonomía de modelos de Deep Learning.

6. Implementación de un modelo de Deep Learning en Python

Existen muchos tipos de capas para constituir distintos modelos de Deep Learning. En este caso, se va a definir un modelo DNN y para ello se utilizarán capas densas y dropout. Las capas densas son el tipo de capa clásico, es decir, capas de redes neuronales donde cada neurona está conectada a las neuronas de la capa anterior y de la siguiente. Cada nodo tiene una función de activación que determina la salida. Las capas dropout son capas que aleatoriamente ponen algunas de las unidades de entrada a 0, reduciendo la posibilidad de overfitting de la red.

El número de neuronas en la capa de entrada deber ser el mismo que el número de características independientes que el banco de datos. Queremos enseñar a la red a reaccionar ante estas características.

La capa de salida, el número de neuronas depende de su objetivo. Los modelos de clasificación tendrían un número de neuronas de salida igual al número de etiquetas de clases diferentes en el conjunto. Para regresión, el modelo deberá ser configurado con una sola neurona de salida.

```
import matplotlib.pyplot as plt
from matplotlib import ticker
import numpy as np
import pandas as pd
import seaborn as sns
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import  random, time, numpy
from sklearn.preprocessing import StandardScaler
```

```
seed=random.seed(time.time())
filename = '../datasets/housing.csv'
col_names=['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',
'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
data = pd.read_csv(filename, names=col_names,sep=" ")

X = data[data.columns[:-1]]
Y = data['MEDV']
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size =
0.3, random_state=seed)

scaler =StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test= scaler.transform(X_test)

model = keras.Sequential([
    layers.Dense(64, activation='relu',
input_shape=[X_train.shape[1]]),
    layers.Dropout(0.3, seed=seed),
    layers.Dense(128, activation='swish'),
    layers.Dense(128, activation='relu'),
    layers.Dense(128, activation='swish'),
    #layers.Dropout(0.2, seed=seed),
    layers.Dense(128, activation='relu'),
    #layers.Dropout(0.1, seed=seed),
    layers.Dense(128, activation='swish'),
    layers.Dense(1)
])

optimizer = tf.keras.optimizers.RMSprop(learning_rate=0.0001)

model.compile(loss=tf.keras.losses.MeanSquaredError(),
              optimizer=optimizer,
              metrics=['mse'])

history = model.fit(
    X_train, y_train,
    epochs=100, validation_split=0.4
)

model_history = pd.DataFrame(history.history)
model_history['epoch'] = history.epoch

loss, mse = model.evaluate(X_test, y_test, verbose=0)
print('RMSE = {:.2f}'.format(numpy.sqrt(mse)))
```

7. PUNTOS CLAVE

En esta lección se ha aprendido:

- Estudiar qué es el Deep Learning y en qué situaciones puede ser útil.
- Reconocer los requisitos software y hardware para desarrollar proyectos de Deep Learning.
- Explicar los distintos modelos de Deep Learning existentes.
- Utilizar las técnicas de implementación de modelos de Deep Learning en Python.